Estruturas de Dados - ESP412

Profª Ana Carolina Sokolonski

Bacharelado de Sistemas de Informação
Instituto Federal de Ciência e Tecnologia da Bahia
Campus de Feira de Santana

carolsoko@ifba.edu.br

November 14, 2024

Estruturas de Dados

Introdução aos Algoritmos Computacionais

Algoritmo:

Sequência finita de instruções, bem definidas e não-ambíguas, para resolver um dado problema. Cada instrução de um algoritmo deve ser executada por um período de tempo finito.

Introdução aos Algoritmos Computacionais

Algoritmo:

Sequência finita de instruções, bem definidas e não-ambíguas, para resolver um dado problema. Cada instrução de um algoritmo deve ser executada por um período de tempo finito.

Algoritmos Computacionais:

São algoritmos que descrevem instruções a serem executadas pelo computador para resolver um problema.

Introdução aos Algoritmos Computacionais

Portanto:

Quando elaboramos um algoritmo computacional devemos especificar ações claras e precisas, que a partir de um estado inicial, após um período de tempo finito, produza um estado final previsível e bem definido.

Introdução aos Algoritmos Computacionais

Portanto:

Quando elaboramos um algoritmo computacional devemos especificar ações claras e precisas, que a partir de um estado inicial, após um período de tempo finito, produza um estado final previsível e bem definido.

Assim, o algoritmo cria um padrão de comportamento a ser seguido, uma norma de execução a ser trilhada, com vistas a alcançar, como resultado final, a solução de um problema, garantindo que sempre que executado, sob as mesmas condições, produza o mesmo resultado.

Introdução aos Algoritmos Computacionais

Entrada:

Conjunto de dados que deve ser fornecido ao algoritmo.

Saída:

Conjunto de dados produzidos pelo algoritmo.

Algoritmos computacionais (Processamento):

Agentes transformadores de dados de entrada em dados de saída.

Introdução aos Algoritmos Computacionais

Entrada:

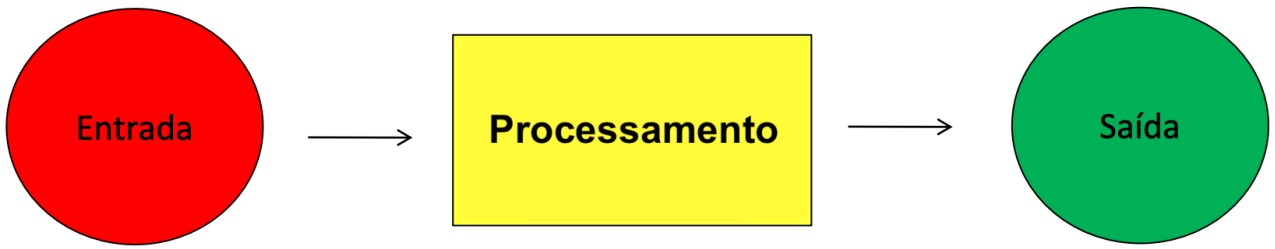
Conjunto de dados que deve ser fornecido ao algoritmo.

Saída:

Conjunto de dados produzidos pelo algoritmo.

Algoritmos computacionais (Processamento):

Agentes transformadores de dados de entrada em dados de saída.



Introdução aos Algoritmos Computacionais

Importante:

Um algoritmo não é a solução de um problema, mas sim um processo para se obter a solução.

Introdução aos Algoritmos Computacionais

Importante:

Um algoritmo não é a solução de um problema, mas sim um processo para se obter a solução.

Problema Computacional:

Pode ser resolvido por um computador através de um algoritmo. Um algoritmo deve ser construído para resolver todas as possíveis ocorrências de um problema.

Introdução aos Algoritmos Computacionais

Importante:

Um algoritmo não é a solução de um problema, mas sim um processo para se obter a solução.

Problema Computacional:

Pode ser resolvido por um computador através de um algoritmo. Um algoritmo deve ser construído para resolver todas as possíveis ocorrências de um problema.

Corretude:

Um algoritmo é dito correto se ele sempre termina e produz a resposta correta para todas as ocorrências de um dado problema.

Introdução às Estruturas de Dados Simples

Introdução às Estruturas de Dados

Estruturas de Dados:

Servem para organizar os dados de modo que eles possam ser processados mais eficientemente. Eficiência tem a ver com Escalabilidade (tempo de processamento):

Introdução às Estruturas de Dados

Estruturas de Dados:

Servem para organizar os dados de modo que eles possam ser processados mais eficientemente. Eficiência tem a ver com Escalabilidade (tempo de processamento):

- Como o tempo de processamento cresce quando a quantidade de dados (ou seja, o tamanho) do problema aumenta?
- O tempo cresce de maneira moderada?
- Cresce de maneira explosiva?

Introdução às Estruturas de Dados

Exemplo: Qual a diferença entre encontrar uma palavra em uma lista de 10 palavras e encontrar esta palavra em uma lista de 10000 palavras? O tempo de processamento é apenas dez vezes maior? É mil vezes maior? Ou é um milhão de vezes maior?

Vetores

Vetores

Definição:

Um *vetor*, ou *arranjo (array)*, é uma estrutura de dados que armazena uma sequência de objetos, todos do mesmo tipo, em posições consecutivas da memória RAM (*random access memory*) do computador. Essa estrutura permite acesso aleatório: qualquer elemento do vetor pode ser alcançado diretamente, sem passar pelos elementos anteriores (o décimo elemento, por exemplo, pode ser alcançado sem que seja necessário passar antes pelo primeiro, segundo, etc., nono elementos). [Cormen et al. 2009]

Vetores

Problemas:

- Como procurar um determinado objeto em um vetor?
- Como inserir um novo objeto no vetor?
- Como remover um elemento do vetor?

Vetores

Problemas:

- Como procurar um determinado objeto em um vetor?
- Como inserir um novo objeto no vetor?
- Como remover um elemento do vetor?

Esses três problemas (busca, inserção e remoção) reaparecerão, em outras estruturas de dados.

Vetores

Suponha que deseja-se armazenar N números inteiros num vetor V. O espaço ocupado pelo vetor na memória pode ser criado pela declaração:

int $v[N]$;

Vetores

Suponha que deseja-se armazenar N números inteiros num vetor V. O espaço ocupado pelo vetor na memória pode ser criado pela declaração:

int $v[N]$;

sendo N uma constante. Considerando que os números são armazenados nas posições de 0 a $(n-1)$, então:

$v[0..(n-1)]$ é um vetor de inteiros.

Claro que $0 \leq n \leq N$. Se $n == 0$, o vetor $v[0..(n-1)]$ está vazio. Se $n == N$, o vetor está cheio.

Vetores

Exercícios:

1. Diga (sem usar o computador) qual o conteúdo do vetor v depois das seguintes instruções:
   int $v[99]$;
   for $(i = 0; i < 99; ++\text{i})$ $v[i] = 98 - i$;
   for $(i = 0; i < 99; ++\text{i})$ $v[i] = v[v[i]]$;

2. Inversão. Suponha que um vetor $v[1..n]$ contém uma permutação de $[1..n]$. Escreva uma função que inverta essa permutação: se $v[i] == j$ no vetor original, queremos ter $v[j] == i$ no fim da execução da função.

RESPOSTAS:

1. O trecho de código copia os números sequenciais de 0 a 49 e depois decresce até 0, desta forma:

$v[0] = 0$
$v[1] = 1$
...
$v[48] = 48$
$v[49] = 49$, a partir daqui começa a decrescer
$v[50] = 48$
$v[51] = 47$
...
$v[98] = 0$

RESPOSTAS:

1. Para visualização, pode-se implementar o algoritmo abaixo:

```c
#include <stdio.h>
#include <stdlib.h>
int main (void){
int i;
int v[99];
for (i = 0; i < 99; ++i)
  v[i] = 98 - i;
for(i=0; i< 99; ++i)
  printf("v[%d] = %d\n", i,v[i]);
for (i = 0; i < 99; ++i)
  v[i] = v[v[i]];
for(i=0; i< 99; ++i)
  printf("v[%d] = %d\n", i,v[i]);
}

```

RESPOSTAS:

O algoritmo de permuta pode ser feito da seguinte forma:

```c
#include <stdio.h>
#include <stdlib.h>
int main (void){
int i,j,aux;
int v1[10],v[]={8,3,4,7,6,5,1,2,9,0};


for (i = 0; i < 10; ++i){
  printf("v[%d] = %d\n", i, v[i]);
  v1[v[i]] = i;
}

  for (j = 0; j < 10; ++j)
    printf("Permutado v[%d] = %d\n", j, v1[j]);

  for (i = 0; i < 10; ++i)
    printf("Original v[%d] = %d\n", i, v[i]);
}
```

Vetores

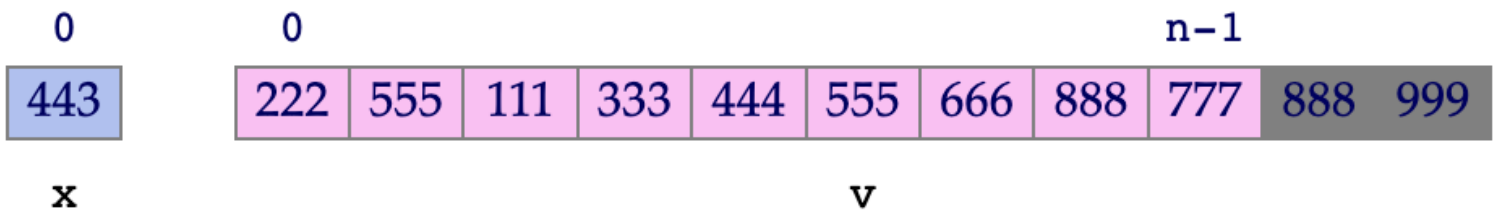
Busca:

Dado um inteiro x e um vetor $v[0..(n-1)]$ de inteiros, o problema da busca consiste em encontrar x em um vetor, ou seja, encontrar um índice k tal que $v[k] == x$.

Vetores

Busca:

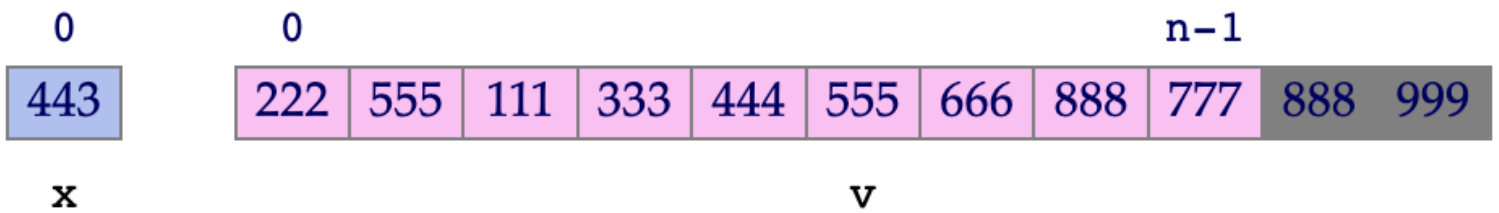
Dado um inteiro x e um vetor $v[0..(n-1)]$ de inteiros, o problema da busca consiste em encontrar x em um vetor, ou seja, encontrar um índice k tal que $v[k] == x$.



Vetores

Busca:

Dado um inteiro x e um vetor $v[0..(n-1)]$ de inteiros, o problema da busca consiste em encontrar x em um vetor, ou seja, encontrar um índice k tal que $v[k] == x$.



É preciso começar com uma decisão de projeto:

- O que fazer quando o problema não tem solução?
- Ou seja, o que fazer se x não está no vetor?

Vetores

Busca:

Pode-se adotar uma convenção, ex. devolver -1 nesses casos. Essa convenção é apropriada, pois -1 não pertence ao conjunto $0..(n-1)$ de índices válidos. Para tanto, deve-se percorrer o vetor em ordem decrescente:

Vetores

Busca:

Pode-se adotar uma convenção, ex. devolver -1 nesses casos. Essa convenção é apropriada, pois -1 não pertence ao conjunto $0..(n-1)$ de índices válidos. Para tanto, deve-se percorrer o vetor em ordem decrescente:

```
int busca (int x, int n, int v[]){
    int i;
    i = n-1;
    while (i >= 0 && v[i] != x)
        i -= 1;
    return i;
}
```

Vetores

Busca:

Este algoritmo está correto mesmo quando $n = 0$. Este mesmo algoritmo pode ser feito ainda menor com um FOR. Ainda mais eficiente e elegante.

Vetores

Busca:

Este algoritmo está correto mesmo quando $n = 0$. Este mesmo algoritmo pode ser feito ainda menor com um FOR. Ainda mais eficiente e elegante.

```
int busca (int x, int n, int v[]){
    int i;
    /**i = n-1;
    while (i >= 0 && v[i] != x)
        i -= 1;**/
    for(i = (n - 1); ((i >= 0) && (v[i] != x)); --i);
    return i;
}
```

Vetores

Exercícios:

1. Escreva uma função que receba x, v e n e devolva 1 se x está em v[0..n-1] e 0 em caso contrário.
2. Suponha que o vetor v[0..n-1] tem dois ou mais elementos iguais a x. Qual deles será apontado pela função busca?
3. Quantas comparações entre x e elementos do vetor v a função busca faz?

Referências

Referências


CORMEN, T. H. et al. *Introduction to Algorithms*. 2nd. ed. [S.l.]: The MIT Press, 2009. ISBN 0262032937.